

SP SciComp '99

October 4–8, 1999

IBM T.J. Watson Research Center

Mixed-Mode Programming

David Klepacki, Ph.D.

IBM Advanced Computing Technology Center

Outline

- MPI and Pthreads
- MPI and OpenMP
- MPI and Shared Memory

Why Mixed-Mode?

- To optimize performance on "mixed-mode" hardware like the SP.
- MPI is used for "Inter-node" communication, and threads (OpenMP / Pthreads) are used for "Intra-node" communication.
 - threads have lower latency and can speed up a code when MPI becomes latency bound.
 - threads can alleviate network contention of a pure MPI implementation.
- To optimize resource utilization (I/O).

MPI and Pthreads

- Rule 1: As long as the same thread performs both the MPI_INIT() and MPI_FINALIZE() functions:
 - pthread functions can be called within the MPI environment, and
 - MPI functions can be called by any pthread.
- Hint: let one thread do all of your MPI work.

MPI and Pthreads

- Rule 2: Multiple threads must execute MPI collective communication operations in the same order on any given communicator.
- Hint: Create a separate communicator for each thread with the `MPI_COMM_DUP()` routine.

MPI and Pthreads

- Thread scope:
 - Prior to PSSP 3.1, you must create pthreads with system contention scope.
 - Beginning with PSSP 3.1, all user pthreads are converted to system contention scope when it makes its first MPI call.

MPI and Pthreads

- When a thread forks a child process (via fork()):
 - only the thread that forked exists for the child task,
 - the forked child should terminate with exit(),
 - if the parent terminates before the child, POE will not clean up.
- A forked child must NEVER call MPI.

MPI and Pthreads

- Environment Variables:
 - Pthreads Variables:
 - SPINLOOPTIME
 - YIELDLOOPTIME
 - XL Compiler Variables (enabled with -qsmp):
 - XLSMPOPTS="spins=100:yields=10:delays=500"
 - The Pthreads and Compiler Variables can interact with each other, so be consistent.

MPI and Pthreads Example

```
#include "mpi.h"
#include <pthread.h>
#include <malloc.h>
typedef struct
{
    double    *a;
    double    *b;
    double    sum;
    int      veclen;
    int      numthrds;
} DOTDATA;
#define MAXTHRDS 8
#define VECLEN 100
DOTDATA dotstr;
pthread_t callThd[MAXTHRDS];
pthread_mutex_t mutexsum;
void* dotprod(void *arg)
{
    int i, start, end, mythrd, len, numthrds, myid;
    double mysum, *x, *y;
/* Each thread does work on a vector of length
VECLEN.*/
```

```
mythrd = (int)arg;
MPI_Comm_rank (MPI_COMM_WORLD, &myid)
numthrds = dotstr.numthrds;
len = dotstr.veclen;
start = myid*mythrd*len + mythrd*len;
end   = start + len;
x = dotstr.a;
y = dotstr.b;
/* Perform the dot product and assign result
to the appropriate variable in the structure.*/
mysum = 0;
for (i=start; i<end ; i++)
    mysum += (x[i] * y[i]);
/*Lock a mutex prior to updating the value in the
structure, and unlock it upon updating.*/
pthread_mutex_lock (&mutexsum);
dotstr.sum += mysum;
pthread_mutex_unlock (&mutexsum);
pthread_exit((void*)0);
}
```

MPI and Pthreads Example

```
void main (int argc, char* argv[])

int i,len=VECLEN, myid, procs, nump1, nthrds;
double *a, *b, nodesum, allsum;
pthread_attr_t attr;
MPI Initialization */
MPI_Init (&argc, &argv);
MPI_Comm_size (MPI_COMM_WORLD, &procs);
MPI_Comm_rank (MPI_COMM_WORLD, &myid);
Assign storage and initialize values */
numthrds=MAXTHRDS;
a = (double*) malloc (procs*nthrds*len*8);
b = (double*) malloc procs*nthrds*len*8);
for (i=0; i<len*procs*nthrds; i++)
{ a[i]=1, b[i]=a[i]; }
dotstr.veclen = len;
dotstr.a = a, dotstr.b = b, dotstr.sum = 0;
dotstr.numthrds=MAXTHRDS;
/* Create thread attribute to specify that the main
** thread needs to join with the threads it creates. */
pthread_attr_init(&attr);
```

```
pthread_attr_setdetachstate(&attr,
                           PTHREAD_CREATE_UNDETACHED)
/* Create a mutex */
pthread_mutex_init (&mutexsum, NULL);
/* Create threads within node for dotproduct */
for(i=0;i<nthrds;i++)
    pthread_create( &callThd[i], &attr,
                    dotprod, (void *)i);
/* Release the thread attribute handle */
pthread_attr_destroy(&attr );
/* Wait on the other threads within this node */
for(i=0;i<nthrds;i++)
    pthread_join( callThd[i], (void **)&status);
nodesum = dotstr.sum;
/* After the dot product, sum results on each node */
MPI_Reduce (&nodesum, &allsum, 1, MPI_DOUBLE
MPI_SUM, 0, MPI_COMM_WORLD)
if (myid == 0) printf ("Allsum = %f \n", allsum);
MPI_Finalize(), free(a), free(b), free(dotstr);
pthread_mutex_destroy(&mutexsum);
exit (0);
```

MPI and OpenMP

- Basic strategy:
 - Decompose problem with MPI,
 - Then add OpenMP.
- Debug and tune MPI and OpenMP separately.

MPI and OpenMP

- Rule: Do not use MPI within an OpenMP parallel region.
 - All MPI calls should be outside any parallel regions or executed by the master thread (OMP MASTER).
 - OMP SINGLE is dangerous; don't use it to call MPI.
- Tip: Use `omp_set_num_threads()` within each MPI task (environment variable `OMP_NUM_THREADS` is implementation dependent, ie, not portable).

MPI and OpenMP Example

```
#include "mpi.h"
#include <malloc.h>
typedef struct
{ double    *a;
  double    *b;
  double    sum;
  int      veclen;
  int      numthrds;
} DOTDATA;
#define MAXTHRDS 8
#define VECLEN=100
DOTDATA dotstr;
void* dotprod()
{ int i, start, end, len, mythrd, numthrds, myid;
  double *x, *y;
/* Each thread does work on a vector of length
   VECLEN.*/
```

```
MPI_Comm_rank (MPI_COMM_WORLD, &myid);
numthrds = dotstr.numthrds;
len = dotstr.veclen;
start = myid*numthrds*len;
end  = start + numthrds*len;
x = dotstr.a;
y = dotstr.b;
/* Perform the dot product and assign result
   to the appropriate variable in the structure. */
dotstr.sum = 0;
#pragma omp parallel for reduction(+:dotstr.sum)
for (i=start; i<end ; i++)
  dotstr.sum += (x[i] * y[i]);
}
```

MPI and OpenMP Example

```
void main (int argc, char* argv[])

int i,len=VECLEN, myid, procs, nthrds;
double *a, *b, nodesum, allsum;
MPI Initialization */
MPI_Init (&argc, &argv);
MPI_Comm_size (MPI_COMM_WORLD, &procs);
MPI_Comm_rank (MPI_COMM_WORLD, &myid);
Assign storage and initialize values */
omp_set_num_threads(MAXTHRDS);
nthrds=MAXTHRDS;
a = (double*) malloc (procs*nthrds*len*8);
b = (double*) malloc procs*nthrds*len*8);
for (i=0; i<len*procs*nthrds; i++)
{ a[i]=1, b[i]=a[i]; }
dotstr.veclen = len;
dotstr.a = a, dotstr.b = b, dotstr.sum = 0;
dotstr.numthrd
```

```
dotprod();
nodesum = dotstr.sum;
/* After the dot product, sum results on each node */
MPI_Reduce (&nodesum, &allsum, 1, MPI_DOUBLE
MPI_SUM, 0, MPI_COMM_WORLD)
if (myid == 0) printf ("Allsum = %f \n", allsum);
MPI_Finalize(), free(a), free(b), free(dotstr);
exit (0);
}
```

MPI Versus OpenMP On-Node

Matrix-Vector Product on beta hardware (8-cpu node):
MP_SHARED_MEMORY=yes

	1 thread	2 threads	4 threads	8 threads
1 MPI task	207.2	105.7	58.4	32.2
2 MPI tasks	105.0	53.5	30.0	
4 MPI tasks	53.5	28.5		
8 MPI tasks	28.4			

Bind Processor Problem

- With current xlf6.1, thread 0 of each MPI task binds to processor 0.
- Performance degradation for multiple MPI tasks per node.
- This problem is will be fixed in future xlf releases.
- Several workarounds available (e.g., Tim Kaiser/Giri Chukkapalli routine from SDSC).

MPI and Shared Memory

- Unix System V IPC can be used to create shared memory regions for use with multiple MPI tasks on a SMP node.
- Private (local) pointers can be used by each MPI task on disjoint subsets of the memory region for easier "data distribution" (but watch out for false sharing).
- A pointer variable can also be used to perform "message-passing" by exchanging only the pointer, and not moving the data (ie, saving a copy).

MPI and Shared Memory

- Headers:

```
#include <sys/types.h>, <sys/ipc.h>, <sys/shm.h>
```

- To allocate "size" bytes,
int shmid, size;
shmid=shmget(IPC_PRIVATE,size,0666|IPC_CREAT);
- A pointer is then used to access this memory,
char *ptr;
ptr = shmat(shmid, 0, 0);
- Cleanup (frees shm resources),
struct shmid_ds *buf;
shmctl (shmid,IPC_RMID,buf);

Shared Memory Example

```
#include <sys/types.h>
#include <sys/ipc.h>
#include <sys/shm.h>

main() {
    int i, shmid, size=8000;
    double *vector;
    struct shmid_ds *buf;

    shmid=shmget(IPC_PRIVATE,size,0666|IPC_CREAT);
    vector=(double *)shmat(shmid,0,0);

    for (i=0; i<100; i++)  vector[i] = i;

    MPI_Init();
    ... MPI, Pthreads, OpenMP ...

    shmctl(shmid,IPC_RMID,buf); }
```